Vectorization Efficiency Metrics

Zakhar A. Matveev, CJ Newburn

with input from Dmitry Prohorov, Hideki Saito

Metrics list

Vectorization Metrics

- Actual speed-up (could be: wall-clock, total, inclusive/exclusive):
 - S = Time (Scalar Loop) / Time (Vector Loop)
 - Efficiency = G/Max_S , Max_S <= Max_VL</p>
- Gain/Efficiency:
 - G = Scalar Loop Cost (cycles) / Vector Loop Cost (cycles)
 - Efficency = G/Max_Gain , Max_G <= Max_VL</p>

How much faster the vector iterations are? Reflects quality of compiler codegeneration.

Should be equal to **S** in case of VPU-bound codes



Vectorization Metrics

Path Reduction:

Scalar Loop Path (# instructions) / Vector Loop Path (# instructions)

Gives a sense of the fraction of non-vector (overhead) instructions in a loop

Vector Utilization / Intensity ("elements active") – only works on KNC

VPU_ELEMENTS_ACTIVE / VPU_INSTRUCTIONS_EXECUTED

Fraction of vector instructions that do work on vector registers. Reflect vector registers utilization

If a mask bit is set for an element, it was presumed to be used. Drops for branchy if-else codes



Intensity/Utilization

Advantage:

Dynamically "measures" fraction of vector instructions that do work (mask-aware)

Disadvantages:

- Only available on KNC (<u>other</u> metrics could be computed on IVB or Broadwell)
- More work/utilization doesn't mean more speed-up (if you care)
 - Some code may have good vector utilization, but scalar version could be faster than it!
 - Shifts/shuffles/"misc.", prefetch instructions are counted "inappropriately".
- Assumption: Mask bits are only set for elements in which useful work is done
 - This is false: Extra mask bits can be set, as long as there are no side effects.
- This is per binary loop, so separate values for peel/remainder..



Gain/Efficiency Estimate

Advantage:

Accounts all nuances of vector vs. scalar assembly and maps it to speed-up

Disadvantages:

- Usually not available if you program in assembly/intrinsics.
- This is code generation performance model, not measurement. This doesn't account dynamic mask *values* as well as other dynamic data (trip counts).

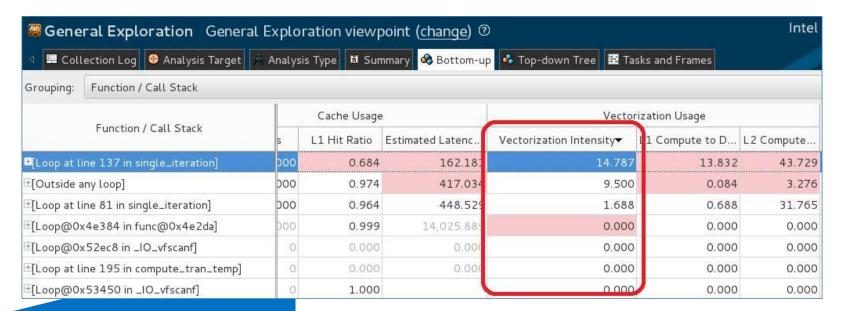
Advisor Gain/Efficiency:

- Recalculate (calibrate) Compiler Gain/Efficiency taking into account dynamic knowledge of trip counts, peel-remainder times.
- But limited to Xeon right now.



Tools to calculate metrics

Intensity/Utilization: VTune Amplifier XE 2016 Beta for KNC



Part of Intel Parallel Studio XE 2016

VTune: General Exploration Analysis Type

Gain/Efficiencty: Intel Compiler (>=15.x)

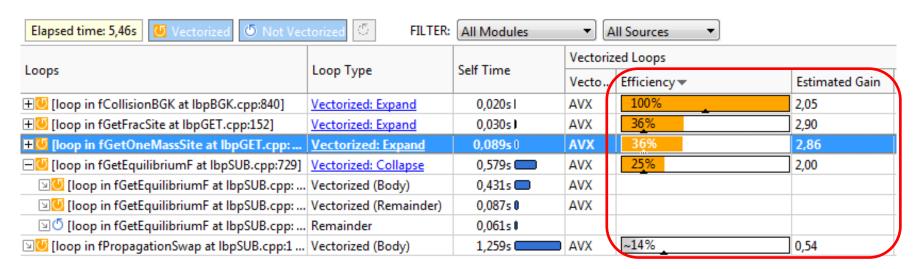
```
OOP BEGIN at C:\work\!LBZ\DL_MESO_LBE 2.6 - Copy\DL MESO_LBE 2.6 - Copy\lbpSUB.cpp(1090,7)
 remark #25408: memset generated
 remark #15542: loop was not vectorized: inner loop was already vectorized
 LOOP BEGIN at C:\work\!LBZ\DL MESO LBE 2.6 - Copy\DL MESO LBE 2.6 - Copy\lbpSUB.cpp(1090,7)
 <Peeled>
    remark #25015: Estimate of max trip count of loop=12
 LOOP END
 LOOP BEGIN at C:\work\!LBZ\DL MESO LBE 2.6 - Copy\DL MESO LBE 2.6 - Copy\lbpSUB.cpp(1090,7)
    remark #15388: vectorization support: reference pt2 has aligned access [ C:\work\!LBZ\DL MESO LBE 2.6
    remark #15305: vectorization support: vector length 4
    remark #15399: vectorization support: unroll factor set to 2
    remark #15300: LOOP WAS VECTORIZED
    remark #15442: entire loop may be executed in remainder
    remark #15449: unmasked aligned unit stride stores: 1
    remark #15475: --- begin vector loop cost summary ---
    remark #15476: scalar loop cost: 4
    remark #15478: estimated potential speedup: 1.110
    remark #15488: --- end vector loop cost summary ---
    remark #25015: Estimate of max trip count of loop=1
 LOOP END
 LOOP BEGIN at C:\work\!LBZ\DL MESO LBE 2.6 - Copy\DL MESO LBE 2.6 - Copy\lbpSUB.cpp(1090,7)
 <Remainder>
    remark #25015: Estimate of max trip count of loop=12
 LOOP END
OOP END
```

Intel Compiler:

-O2 -opt-report5

Part of Intel Parallel
Studio XE 2016

Gain/Efficiencty: Advisor XE 2016 Beta (for Xeon only)



Part of Intel Parallel
Studio XE 2016

Advisor: Survey Analysis Type

Some word on methodology...

WHAT to measure?

Actual Speed-up vs. Efficiency vs. Intensity. ??

- Measuring all and comparing results –
 is "useful enough" exercise already.
- Normally stick with at least one of them for workshop exercises.

Kernel vs. Sub-part vs. Workload ??

- Per-workload speed-up/efficiencies are lower than per-kernel (Amdahl's law)
- Both are important to understand, but don't mix them up!
- For big HPC codes you rarely even look into everything. Define sub-set.
- Measuring / establishing proper baselines is very important/not-trivial itself



Some take-aways

Vectorization efficiency/gain

- Take it as input, but treat it as performance estimate
- Use Advisor if you want to overcome some of "static code-generation knowledge" limits

Vectorization intensity

- Take it as input, but don't treat it as accurate:
 - Low intensity definitely means you have an issue. Otherwise who knows.
- If higher than expected, inspect code for masks that are all 1 even through conditionals
- The VPU_ELEMENTS_ACTIVE won't be available for *anything* other than KNC

Don't compare apples with oranges (kernel and workload, etc)

Don't mix up dimensional and non-dimensional metrics

